University of Saskatchewan
Department of Computer Science

Cmpt 330
Midterm Examination

October 27, 2003

Time: 50 minutes
Total Marks: 63

Professor: A. J. Kusalik
Closed Book

Name: _____

Student Number: _____

Directions:

Answer each of the following questions in the space provided in this exam booklet. If you must continue an answer (e.g. in the extra space on the last page, or on the back side of a page), make sure you clearly indicate that you have done so and where to find the continuation.

Make all written answers legible; no marks can be given for answers which cannot be decrypted. Where a discourse or discussion is called for, please be concise and precise. Do not give "extra answers". Extra answers which are incorrect may result in your being docked marks.

Use of calculators is not allowed during the exam. Fortunately, you should not need a calculator for completing any of the questions.

The context for questions is the UNIX operating system, as manifest by NetBSD, unless explicitly stated otherwise. If you find it necessary to make any assumptions to answer a question, state the assumption with your answer.

Marks for each major question are given at the beginning of that question. There are a total of 63 marks (one mark per minute).

Good luck.

For marking use only:

A. _____/11	D. _____/13	G. _____/6
B. _____/9	E. _____/6	H. _____/3
C. _____/6	F. _____/9	
		Total: _____/63

A. ***(1 mark each for 11 marks total)***

For each of the statements below, indicate whether it is **true** ("T") or **false** ("F").

___ An operating system kernel (plus the underlying hardware) can be thought of as a virtual machine.

___ A system call (or kernel call) is a portion of user code (code in the user's address space) which is executed by the CPU with the processor status word (PSW) set to "kernel" or "privileged" mode.

___ A process goes through a number of states as it proceeds through its computation. Three of the possible states are *Runnable*, *Sleeping*, and *Zombie*.

___ Environment variables are stored in the "u dot" area of a process.

___ If a specific signal is delivered to a process and a signal handler initiated, further signals of that type (that signal number) are blocked until the handler completes (e.g. executes a `return` statement).

___ The construct for handling signals in an *sh* script is called a "here document".

___ The superblock (of a UNIX file system) is always at datablock 16 of the underlying hard disk storage.

___ UNIX allows for dynamic mounting and dismounting (or unmounting) of removable file systems. These file systems can come from a variety of sources, including local disks and remote file servers.

___ The *dup(2)* kernel call is used to duplicate a file; i.e. duplicate an i-node.

___ The following fragments of C code (which are the sort of code one would expect to find in a *.h* file) comes from `<stdio.h>` (i.e. `/usr/include/stdio.h`):

```
#define BBSIZE          8192
#define SBSIZE          8192
#define BBOFF           ((off_t)(0))
#define SBOFF           ((off_t)(BBOFF + BBSIZE))
#define BBLOCK          ((daddr_t)(0))
#define SBLOCK          ((daddr_t)(BBLOCK + BBSIZE / DEV_BSIZE))

#define FS_MAGIC        0x011954
                          /* the fast filesystem magic number */
#define FS_OKAY         0x7c269d38      /* superblock checksum */
#define FS_42INODEFMT   -1              /* 4.2BSD inode format */
#define FS_44INODEFMT   2               /* 4.4BSD inode format */

#define fsbtodb(fs, b)  ((b) << (fs)->fs_fsbtodb)
#define dbtofsb(fs, b)  ((b) >> (fs)->fs_fsbtodb)
```

___ A "hole" in a UNIX file is used to reserve space in the file (so that other files cannot use the space).

B. ***(1+1+2+1+1+1+1+1 = 9 marks)***

Answer each of the following questions with a very short answer.

1. What option does one give to the *cc* or *gcc* command so that the C compiler includes, in its output file, symbol table information used by debuggers such as *gdb* and *ddd*?

2. The main "streams" in the evolution of UNIX include the "BSD" and "LINUX" streams. What is a third main "stream"?

3. Consider the following program from the Molay textbook:

```
/* forkdemo2.c - shows how child processes pick up at the return
 *               from fork() and can execute any code they like,
 *               even fork().
 */

main()
{
   printf("my pid is %d\n", getpid() );
   fork();
   fork();
   fork();
   printf("my pid is %d\n", getpid() );
}
```

Assuming that `stdout` is a terminal, display, or window (i.e. so that *`fflush()`* is automatically called as part of the code for *`printf()`*), how many times will the

`my pid is <`*`some number`*`>`

line be printed?

4. What must a text file begin with to be recognized as an executable script?

5. In a UNIX file system, what is the inode number of the root of the file system?

6. Suppose a user types the "*`ls -lgi`*" command and gets the following output:

```
> ls -lgi
total 0
3517 -rw-r--r-- 1 root wheel 0 Jan 23 12:17 a
3527 -rw-r--r-- 1 root wheel 0 Jan 23 12:17 b
3529 -rw-r--r-- 2 root wheel 0 Jan 23 12:17 c
3529 -rw-r--r-- 2 root wheel 0 Jan 23 12:17 d
```

Given the output above, name (give the names of) all the hard links in the user's current working directory.

Note: your answer has to be completely correct to obtain the mark for this question.

7. Consider the fictional (but complete) directory listing below (as would be produced using "*ls -l*"). One value in the listing has been replaced with the symbol **A**. What value should be present in the listing in place of **A**?

```
tonka3# ls -al example
drwxr-xr-x A root 512 Jun 5 15:12 .
drwxr-xr-x 12 root 512 Nov 11 1994 ..
-r--r--r-- 1 bin 5486 Jan 19 1998 dinode.h
-r--r--r-- 1 bin 6114 Apr 28 1998 dir.h
-r--r--r-- 1 bin 6569 Mar 9 1997 inode.h
drwxr-xr-x 2 root 512 Jun 5 15:12 mfs
-rwxr-xr-x 1 root 1 Jul 30 02:51 testsym1
-rwxr-xr-x 1 root 12 Jul 30 02:51 testsym2
drwxr-xr-x 2 root 512 Jun 5 15:12 ufs
```

8. A command that can be given to the *csh(1)* (C-shell) is `csh`. I.e. a child process running *csh* can be initiated from *csh*. Is the command "csh" therefore a built-in command in the C-shell?

C. (2+2+2 = 6 marks)

Suppose your current working directory (for the current instance of your interactive shell) contains the following files (and only these files):

```
log         temp.ps     test2.c
show.sh     test1.c
```

Also assume that your interactive shell is *csh*.

Consider each of the following commands given to the aforementioned interactive shell. In each case, the shell will *fork()* a new child process and *exec()* the specfied command (program) with appropriate arguments. For each command, say how many arguments will be actually specified (given) to the *exec()*-ed program. I.e. in each case, when the appropriate system program is invoked by *exec()*, how many arguments will it have (be given)? For example, for the command

```
ls -l log show.sh
```

ls is invoked with four arguments (the command name, the option "-l", and two filenames).

1. `echo "The files are: \*"`

2. `echo 'The path is $PATH > log' &`

3. `grep pole temp[123].?`

D. (2+2+2+3+1+1+2 = 13 marks)

Answer each of the following questions with a short, but precise descriptive answer.

1. What is the difference between sections 1 and 8 in the UNIX manual set? How does the information you would find in the two sections differ?

2. A user has an *sh* (bourne shell) script with the following content

```
#! /bin/sh
sleep 5
exit
```

stored as file `show1`.

The user's normal shell is *tcsh*. The script is invoked as a command, as in

```
> ./show1
```

This results in the following process structure (as reported by *pstree(1)*):

```
tcsh---sh---sleep
```

where the "tcsh" is the shell to which the user is giving commands. What would the output from *pstree* look like if the shell script were invoked via

```
> sh show1
```

instead?

3. What is the primary difference between the signals `SIGQUIT` and `SIGTERM` (other than their numeric value)?

4. It is possible for a file to have permissions set such that its owner cannot read it, though other users still can. Suppose that this is the case for a file `foobar` owned by user `charlie`. Give a plausible output from the command

```
ls -l foobar
```

which would be indicative of this situation.

5. One of the problems with the *fork+exec* scheme for process spawning is the wastage caused by copying an address space (from parent to child) only to overwrite it (child performing an *exec*). What is one optimization used in UNIX systems to overcome this problem?

6. In UNIX, why are ordinary users never allowed to write (directly) to a directory (a directory file)?

7. How can you redirect the output of the find command (or any UNIX command, for that matter) so that whatever it sends to standard error "disappears" (is not output to the user)? Show below the construction in *sh* which will do this.

E. (3+3 = 6 marks)

For each of the following C code segments, indicate and describe the error being made. I.e. in each of the following pieces of code, a programming error is being made. Identify (in some easily discernible way) the error in each case. Also describe (by way of a short description) the nature of the error. Then indicate how the error can be concisely corrected without changing the intended logic of the program sample. Marks will be deducted for reporting non-errors.

Hint: look for C programming errors as well as errors in usage of kernel calls.

1. The following program is intended to retrieve the datablock size for the file system which is storing the file named on the command line. The datablock information is stored in the statfs structure provided by *statfs()*. Unfortunately, the program aborts with a memory-related error. What is the nature of the error? How would you fix it?

```
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/param.h>
#include<sys/mount.h>

void main(int argc, char *argv[])
{
   /* declaration of variables */
```

```
    struct statfs *block;  /* defined in library file sys/mount.h */
    char *file;
    long size;

    /* If the first argument is not specified use the current
       working directory */
    file = (char *)calloc(16,sizeof(char));
    if (argc == 1)
    {
       file = ".";
    }

    /* Exit if there is more than one file specified */
    else if (argc != 2)
    {
       printf("This program only takes one argument\n");
       exit(1);
    }

    /* Set file or path to argument given on command line */
    else
    {
       file = argv[1];
    }

    /* Copy file's file information into the block variable */

    if ( statfs(file, block) != 0 )
    {
       perror("Error reading file data");
       exit(2);
    }

    /* Retrieve information on datablock size and output it */
    size = block->f_iosize;
    printf("The block size on this mounted file system is: %ld\n",
     size);

    exit(0);
}
```

The man page for *statfs* includes the following portions of information:

```
NAME
     statfs, fstatfs - get file system statistics

SYNOPSIS

     #include <sys/param.h>
     #include <sys/mount.h>
```

```
int
statfs(const char *path, struct statfs *buf);

int
fstatfs(int fd, struct statfs *buf);
```

DESCRIPTION

statfs() returns information about a mounted file system. path is the path name of any file within the mounted file system. buf is a pointer to an existing statfs structure defined as follows:

```
typedef struct { int32_t val[2]; } fsid_t;
                        /* file system id type */

#define MFSNAMELEN   16 /* len of fs type name, including nul */
#define MNAMELEN     90 /* length of buffer for returned name */

struct statfs {
    short   f_type;     /* type of file system (unused; zero) */
    u_short f_oflags;   /* deprecated copy of mount flags */
    long    f_bsize;    /* fundamental file system block size */
    long    f_iosize;   /* optimal transfer block size */
    long    f_blocks;   /* total data blocks in file system */
    long    f_bfree;    /* free blocks in fs */
    long    f_bavail;   /* free blocks avail to non-superuser */
    long    f_files;    /* total file nodes in file system */
    long    f_ffree;    /* free file nodes in fs */
    fsid_t  f_fsid;     /* file system id */
    uid_t   f_owner;    /* user that mounted the file system */
    long    f_flags;    /* copy of mount flags */
    long    f_syncwrites; /* count of sync writes since mount */
    long    f_asyncwrites; /* count of async writes since
                              last mount */
    long    f_spare[1]; /* spare for later */
    char    f_fstypename[MFSNAMELEN]; /* fs type name */
    char    f_mntonname[MNAMELEN];
                        /* directory on which mounted */
    char    f_mntfromname[MNAMELEN];  /* mounted file system */
};
```

RETURN VALUES

Upon successful completion, a value of 0 is returned. Otherwise, -1 is returned and the global variable errno is set to indicate the error.

2. The following fragments of C code are from a program which uses a temporary file for reading and writing . The fragments encompass an error which has been made in the program. What is the error? How can it be fixed?

```
#include <stdio.h>
#include <fcntl.h>
.
.
.
FILE *myfile;
.
```

```
.
.
if( (myfile=open( "/tmp/myfile", O_CREAT|O_TRUNC|O_RDWR, 0640 )) < 0
)
{
    fprintf( stderr, "cannot open temporary file" );
    exit( 1 );
}
```

F. (2+4+3 = 9 marks)

Answer each of the following questions with a focused discussion-oriented answer. You may wish to use examples and/or diagrams to illustrate points in your answer.

1. What is a vulnerability of the superblock in the original UNIX file system?

2. A process can terminate (its own) execution in a number of different ways. One is via the *exit()* library call, as in

```
exit( 3 );
```

Another is by sending itself a signal which causes termination, as in

```
raise( SIGKILL );
```

or

```
abort( );
```

How can a parent determine if a child it has created has terminated via

```
exit( 3 );
```

or

```
abort( );
```

How can the parent process determine the difference? What kernel call or library call would the parent use? What information provided by such calls would the parent process check? What would it look for in that information?

3. One characterization of a UNIX process is that it is a "program running in an address space". In class, it was stated that such a characterization is not completely accurate. How so? How is it that a UNIX is not exactly a "program running in an address space"?

G. (3+3 = 6 marks)

Suppose you are filling out a web-based form. You are to provide a file to be uploaded. However, on the web page is the following warning:

> WARNING: the complete directory path (including file name) must not exceed 64 characters.

Suppose the file to be uploaded is called `report.txt`, and it exists in your current working directory. You would like to check the length of the absolute pathname of the file before entering it on the web form. You therefore open a virtual terminal window with a UNIX shell in it, *cd* to the same working directory, and perform some UNIX commands to check the length of the pathname.

0. State which shell (*sh* or *csh*) you intend to use.

Note: the answer to this first question will not be graded. It is necessary to establish the correctness of the answers to the questions below.

1. Give a simple shell command which will set a user-defined (shell) variable `F` to have, as its value, the full, absolute pathname of the file, `report.txt`.

2. Using the variable `F` in the question above, give a shell command which will determine the length of the full pathname (of file `report.txt`).

H. (3 marks)

A user has the following files in his directory

```
sa3.c sa4.c sa5.c sa6.c sa7.c sa8.c sa9.c
sb1.c sb2.c sb3.c sb4.c sb5.c
```

The user then issues the following command to *sh* (Bourne shell):

```
for i in 3 4 5 6 7 8 9
do
mv -f sa${i}.c sb`expr $i + 4`.c
done
```

What files will exist in the directory after this complex command completes? Give a list of names of all the files in the directory after the above command is executed.